John von Neumann Institute for Computing

A Speculative Parallel Algorithm for Self-Organizing Maps

C. Garcìa, M. Prieto, A. Pascual-Montano

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 615-622, 2006.

http://www.fz-juelich.de/nic-series/volume33

A Speculative Parallel Algorithm for Self-Organizing Maps*

C. García[a], M. Prieto[a], A. Pascual-Montano[a]

[a]Dpto. de Arquitectura de Computadores y Automática
Universidad Complutense, 28040 Madrid, Spain
e-mail: {garsanca, mpmatias}@dacya.ucm.es, pascual@fis.ucm.es

We have explored in this paper the parallel implementation of *Kohonen's Self Organizing Map* (SOM) in simultaneous multithreading architectures. A new method is proposed that outperforms classic *map-partitioning* approaches targeted for shared-memory multiprocessors. As an application study we have chosen an image classification problem in 3D Electron Microscopy. Performance results are taken using real biological data on an *hyperthreading*-enabled Intel Pentium 4.

1. Introduction

Electron Microscopy (EM) is a valuable tool for the elucidation of the three-dimensional structure of macromolecular complexes [3]. However, it faces different methodological problems. Most of the methods used for 3D reconstruction in EM rely on the strict requirement that the individual projection images considered for the reconstruction process, correspond to different views of the same biological specimen. The achievement of such a set of particles makes necessary a preprocessing step aimed at sorting the original population of images into different homogeneous sub-populations. This classification is also difficult since in most of the instances no prior information on the macromolecule structure is available. Furthermore, the large amount of electron microscopy projection images needed for a single study (usually tens of thousands), makes its computational efficiency another major concern for researches.

In this paper we have addressed this efficiency issue, focusing on simultaneous multithreading (SMT) processors and the well-known *Self-Organizing Map* (SOM) algorithm [9]. SOM was introduced in the EM field by Marabini and Carazo in [12], although more recently new variants were proposed [15–17].

As its name suggests, SMT allows several independent threads to issue instructions simultaneously in a single cycle [20]. Its main goal is to yield better use of the processor's resources, hiding the inefficiencies caused by long operational latencies. Some sort of SMT (denoted as *hyperthreading*) has been already incorporated into the Intel's Pentium 4 and Xeon families [13] as well as the IBM's Power5 [7], being expected to become ubiquitous soon in most superscalar processors. Despite this architectural trend, the exploitation of this capability is often limited by the relative underdevelopment of the compilers (i.e. automatic parallelization).

This noticeable divergence between compiler technology and computing power has forced us to explicitly adapt SOM to take advantage of the simultaneous thread-issue capability. At first glance, SMT processors can be seen as a set of logical processors that share some resources. Consequently, one may think that optimizations targeted for symmetric multiprocessors systems (SMP) are also good candidates for SMT. However, we will show that this naïve view does not provides satisfactory speedups.

*This work has been supported by the Spanish research grants TIC 2002-750 and GR/SAL/0653/2004. A.P.M. also acknowledges the support of the Spanish Ramon y Cajal Program.

As a baseline code to study SMT-aware optimizations, we have employed a modified version of the well-known SOM-PAK package from the Helsinki University of Technology [18]. This tuned version already includes some code rearrangement that allows for an efficient exploitation of the Intel's SSE and SSE2 media extensions. Experiments have been performed on an Intel Pentium 4 running at 3.2 Ghz (1MB L2 cache, 1Gb DDR400). Thread synchronization has been performed using POSIX Threads.

The rest of this paper is organized as follows: Section 2 introduces SOM and summarizes its most popular parallel implementations; Section 3 describes the application problems used for validation and assessment. Section 4 presents our alternative implementation and its performance. Finally the paper ends with some conclusions.

2. Self-Organizing Maps

The *Self-Organizing Map* (SOM) is an unsupervised neural network that provides a non-linear mapping from a high-dimensional input space to a low-dimensional output space (most often a 2D output grid). Its simplicity in description and practical implementation, have made SOM one of the most popular and widely used methods for pattern recognition and exploratory data analysis [8]. Many studies have also confirmed its ability in producing an orderly mapping of high dimensional data items onto a regular low dimensional grid. Its main property is that it quite consistently conserves the original topological and metric relationships of the items.

As show in Figure 1, the SOM consists of a set of *i* input units, corresponding to the input data set, and a set of *j* output units arranged in a two-dimensional regular grid with a predefined topology. Each output unit has a codevector $w_i \epsilon \Re^p$ associated with it.

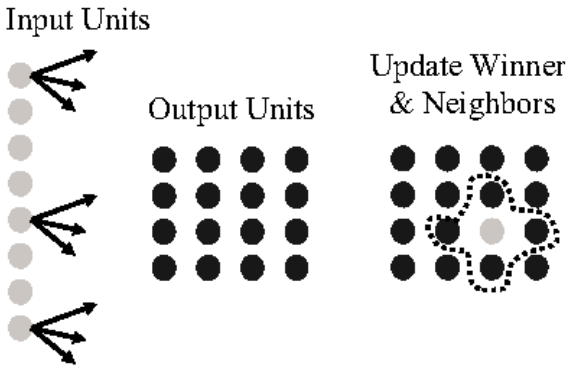

Figure 1. Self-Organizing Maps.

The functionality of the algorithm can be described as follows: when an input vector ($x_i \epsilon \Re^p$) is presented to the net, (1) the neurons or units in the output layer compete with each other and (2) the winner (which is that neuron whose codevector has the minimum distance from x_i) as well as a predefined set of its neighbors, update their values. This process is repeated until some stopping criterion is met, usually, when the codevectors stabilize or when a certain number of iterations are completed. The update rule for the output vectors used in this algorithm can be mathematically described as:

$$w_k(t+1) = w_k(t) + \alpha(t) h_{ck}(t)[x(t) - w_k(t)] \quad (1)$$

Where $\alpha(t)$ is a decreasing function of t (time, or iteration number) that controls the magnitude of the changes with time, and $h_{ck}(t)$ is a function that controls the size of the neighborhood of the winning node to be updated during training. Both $\alpha(t)$ and $h_{ck}(t)$ decrease monotonically during training in order to achieve convergence. This classic algorithm is usually denoted as *on-line*, given that it updates the codevectors at each step [6].

2.1. Conventional Parallel Self-Organizing Maps

A significant amount of work on the parallel implementation of SOM has been performed in recent years [10,1,14,11]. In general, two different alternatives, which we have denoted as *map-partitioning* (MP) and *data-partitioning* (DP), have been explored.

In *map-partitioning*, the SOM is distributed among the different threads so that every thread processes every input data and trains its share of the map. This approach preserves the ordering of updates shown in Eq. (1) and hence produces exact agreement (within round-off error) with the *on-line* algorithm described above. Its main drawback is its high synchronization cost: threads should synchronize at each step after determining the winning output unit, and again after updating the map. This overhead only makes MP attractive for hardware implementations, and for parallel architectures with low latency synchronization primitives such as *Shared Memory Symmetric Multiprocessors* (SMP) or SMT processors.

In *data-partitioning*, input data are distributed across threads so that every thread trains a full copy of the map using a subset of the input space. Its granularity is coarser than the small-grained parallelism available in MP, which decreases synchronization costs. Unfortunately DP does not fit with the *on-line* update established by Eq. (1) and hence it should be combined with different variants of SOM. A typical choice is *batch* SOM [9,6,10,11]. However, *batch* SOM is not very popular in the scientific community because of two main reasons:

1. It requires all data items to be present, which is impractical in some applications due to communication time or disk space requirements.

2. It is more difficult to obtain a properly ordered map than with the classical *on-line* algorithm [6].

Taking both factors into account, and given that a SMT processor provides for efficient thread synchronization, we have used a MP implementation of the classic *on-line* SOM as a point of reference for evaluating the advantages of our alternative.

3. Description of the application problems

The validation and assessment of our method has been performed using three different real datasets corresponding to three different macromolecular studies:

- *MCM*. In this study we used 4723 single particles (64x64 projection images) from the MCM helicase from Methanobacterium Thermoautotrophicum obtained by electron microscopy [5]. The aim of SOM is to study the structural heterogeneity of this macromolecule in an attempt to understand its biological function. Each 64x64 image was arranged in a vector resulting in a 4723x4096 data matrix. Both the serial and the map-partitioning versions of the *on-line* SOM algorithms were applied to this dataset using a 30x10 hexagonal grid.

- *G40P*. This test consists of 2120 EM images corresponding to 2D projections of negatively stained hexamers of the bacteriophage SPP1 G40P helicase [2]. From each image, 1580 pixels

within an area of interest were extracted using a binary mask. The aim of SOM in this case, which is computed using a 2120x1580 data matrix, is to study the structural polymorphism of this macromolecular assembly.

- *Tomograms*. To extend our study to a more demanding case, we also tested our method using 3D images from electron tomography. The problem in this case is focused on classification of three-dimensional volumes, which represent a much larger and complex problem. The dataset used for testing contained 1000 3D images from the insect flight muscle specimen. The tomograms were properly carved out and aligned from the tomographic map as described in [21]. As in *G40P*, only a specific area of interest was extracted using a proper binary mask producing 1000 vectors with 19475 components (voxels) each. More details about this dataset can be found in [21,17].

4. Speculative Parallel SOM

4.1. Motivation

Rows labeled MP in Table 1 show the speedups achieved by *Map-Partitioning*. We also report the speedups achieved on a dual Intel's Xeon server to emphasize the discrepancies between SMT processors and SMP systems. As mentioned above, the baseline code refers to our hand-tuned version of SOM-PAK, in which the computation of distances between input and output units as well as the update of codevectors, have been vectorized using the intrinsic function interface provide by the Intel C/C++ compiler [4]. Vectorization already achieves a significant speedup that ranges between 3 and 4,depending on the size of the images and the number of items.

As expected, MP provides satisfactory results on SMP platforms. However, the corresponding performance gains achieved on SMT are negligible, either using a block or a cyclic distribution of output units across threads.

The main reason behind this poor behavior is the competition among threads for the shared resources, especially for the largest dataset (*Tomograms*). In particular, we have observed that given that both threads process different output units, they must compete for the memory bandwidth and share data caches. Driven by these discouraging results we have devised an alternative way to perform the computation, in which competition among threads for the data caches is much lower.

4.2. Description

Our alternative algorithm is based on the following observations:

1. In the first steps of training, the neighborhood of the winning node of a certain input vector usually overlaps with the neighborhoods of the winning nodes of consecutive input vectors.

2. Given that $h_{ct}(t)$, i.e. the neighborhood radius, decreases monotonically during training, the chances of overlapping also tends to decrease.

3. If the neighborhoods of two consecutive winning nodes do not overlap, it would be possible to process the respective input units in parallel.

Our proposed algorithm tries to exploit the potential parallelism described by the third observation. It uses a master-slave scheme (see Figure 2), in which the master thread basically follows the conventional *online* SOM algorithm, and the slave or helper thread tries to train the map with the successive input units in a speculative way. After determining its winner node (both threads perform

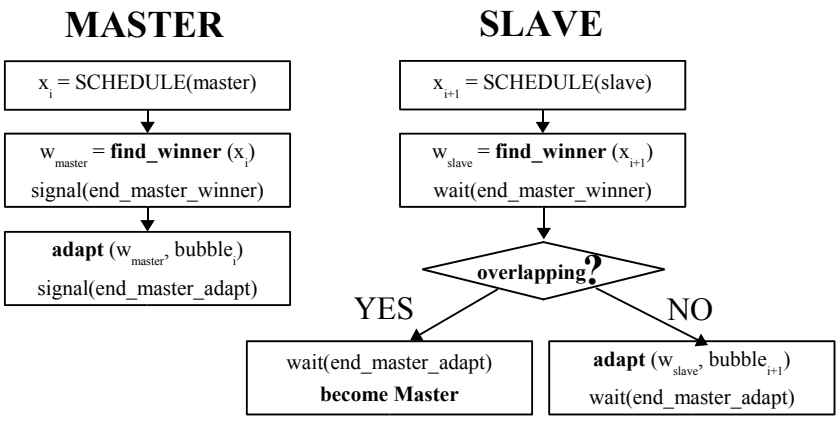

Figure 2. Master-slave scheme.

the *find_winner* stage of the algorithm in parallel), the slave checks if its respective neighborhood overlaps with the master's neighborhood. If they do overlap, the speculation has been wrong and the slave must wait until the end of the master's update phase, and then master and slave exchange their roles. The new slave tries speculatively to process another input vector. If they do not overlap, we assume that both, the master and the slave threads, can update the map concurrently and proceed with the next input vectors.

Since the chances of overlapping are high at the beginning and tend to decrease during training, speculation starts after processing a certain number of input-vectors. Based on our experiments (see Section 4.3), the length of this *warm-up* phase has been set to the first ten percent of the iterations. *Thread parallelism* is also exploited in this initial phase applying MP.

The obvious disadvantage of speculation is that some resources are allocated to useless computations that must be re-executed. However, since resources are often underutilized on current microprocessors [20], the benefit of speculation could far outweigh this disadvantage. It is also worth to note that this scheme can be scaled using more slaves, and sorting them out so that the overlapping checking is performed in order. Nevertheless, within current design trends (current SMT processors only allow for two simultaneous threads), this possibility has no practical interest.

4.3. Performance Results

Rows labeled SP in Table 1 show the speedups achieved by our alternative approach. Obviously, in the dual system its performance is significantly worse than MP but in the SMT processor, it far outperforms MP if vectorization is enabled (baseline code). The results are quite satisfactory taking into account the achievable speedups reported by Intel (around 30% on average)[19]. Hardware performance counters have confirmed that this better behavior is due to its more efficient exploitation of the memory hierarchy. Unlike MP, where threads compete for the data caches and the memory bandwidth, the master and the slave threads tend to cooperate during the *find_winner* stage, since they analyze each of the output units approximately at the same time. This way, temporal locality is better exploited. We should highlight that in the SP model, a certain synergy between SMT and vectorization exists. Vectorization allows a better exploitation of the shared resources and allows for a better cooperation amongst threads. It is also worth to note that the overall speedup of our new implementation over the original SOM-PAK version is close to 5 (on average).

Figures 3-5 show the evolution of the misspeculation rate (the percentage of overlaps found during

Table 1
Speedups achieved by a conventional *Map-Partitioning* decomposition (MP) and the proposed *Speculative* implementation (SP) on a SMT processor and a dual processor server (denoted as SMP) respectively for the three different databases considered.

MCM	SMT	SMP
Original-MP	1.03	1.77
Baseline-MP	1.07	1.59
Original-SP	1.01	1.46
Baseline-SP	1.24	1.08

G40P	SMT	SMP
Original-MP	1.04	1.91
Baseline-MP	1.09	1.89
Original-SP	1.02	1.41
Baseline-SP	1.23	1.12

Tomograms	SMT	SMP
Original-MP	1.04	1.68
Baseline-MP	1.00	1.17
Original-SP	1.01	1.48
Baseline-SP	1.11	1.06

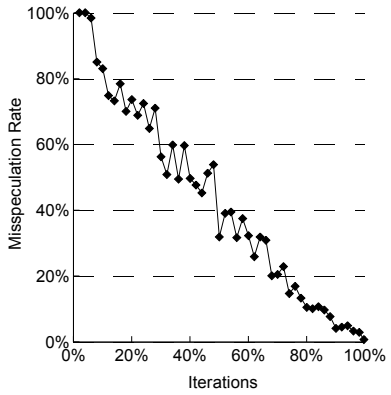

Figure 3. Evolution of the misspeculation rate for the MCM dataset.

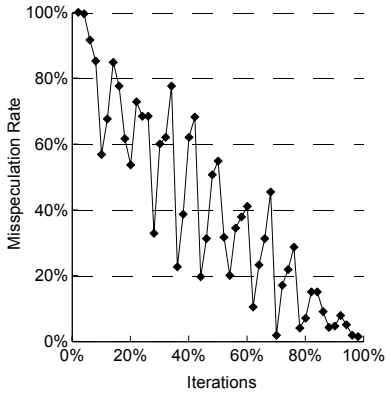

Figure 4. Evolution of the misspeculation rate for the G40P dataset.

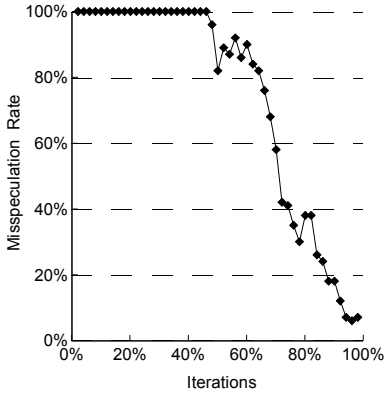

Figure 5. Evolution of the misspeculation rate for the Tomograms dataset.

execution) using the SP model without a *warm-up* phase (i.e. applying speculation from the first iteration). They explain the worse behavior observed on the *Tomagrams* dataset since the number the overlaps keeps high for a large number of iterations. Increasing the warm-up length in this case does not improve performance significantly since it only removes the misspeculation overheads. In fact, these overheads are not very significant since misspeculation also causes data-prefetching as positive side effect.

4.4. Validation

Figure 6 compares the results obtained by the classic *online* SOM algorithm and our speculative approach using as input the *MCM* dataset. The numbers in the lower right corner of each unit represent the number of original images assigned to each code vector in the map. It is visually evident that both maps are almost identical, which represents a qualitative evidence that the speculative approach do not affect the overall numerical output of the map. The correlation coefficient of both maps yielded a 0.9999 similarity. The small differences between both results are only reflected in the number of images represented by each of the code vectors in those maps. In some cases, a few very similar images are assigned to different neighboring units. This is not a problem whatsoever due to the fact that SOMs represents a smooth representation of the input data in the 2D ordered grid and therefore, a cluster of images is determined by a set of neighboring units and not by individuals.

The resulting maps obtained with the *G40P* dataset showed a correlation coefficient of 0.9981 and, again, this difference is only reflected in the location of some similar images that were allocated to neighboring units in the map. Results demonstrate that even if our *Speculative* calculation of SOMs

might introduce some small changes in the code vectors, the overall results are not affected.

Finally, in the case of tomograms dataset, the resulting maps using the *online* and the *speculative* SOMs were identical (correlation coefficient of 1).

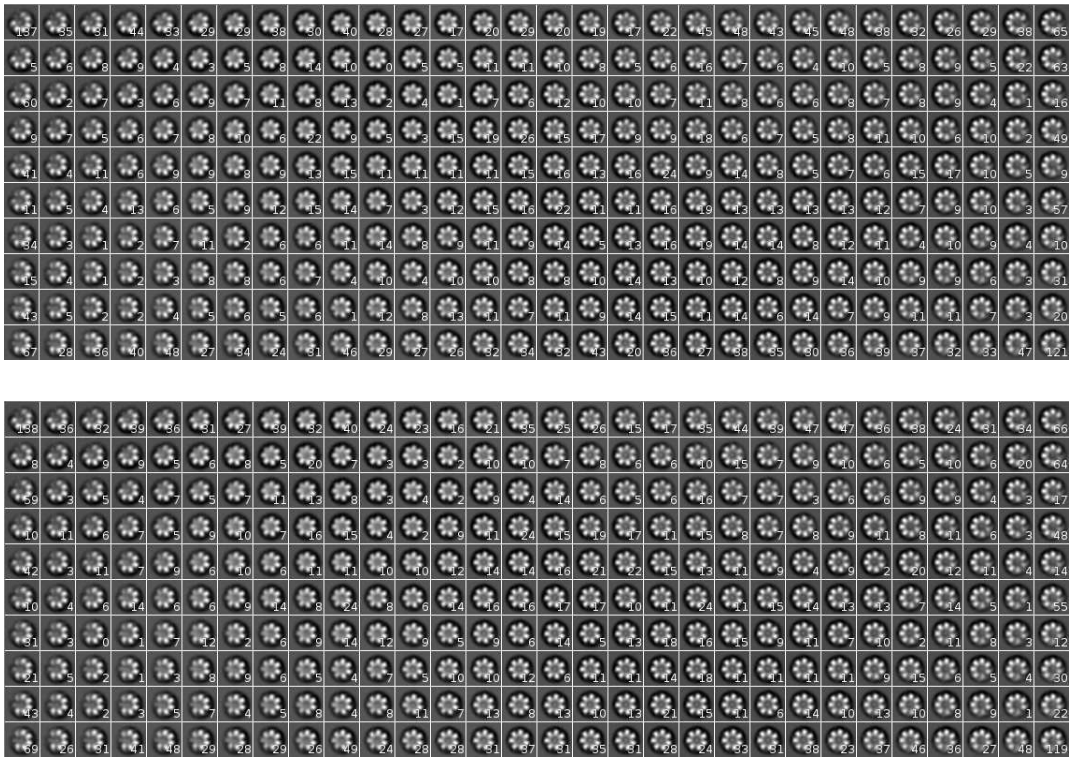

Figure 6. 30x10 Self-Organizing Map obtained using the *online* Kohonen Map (top) and the proposed *speculative* Kohonen Map (bottom).

5. Conclusions

The self-organising map is a popular unsupervised neural network model for high-dimensional data analysis. However, the high execution times required to train the map put a limit to its use in many application domains, where either very large datasets are encountered and/or real-time response times are required.

We have introduced a new parallel implementation of the classic *Self-Organizing Map* algorithm, which on SMT processors fits better than traditional *Map-Partitioning* strategies due to a better exploitation of the memory hierarchy. It is also worth to note the numeric results achieved with this speculative approach are almost identical to those obtained with the serial *online* SOM.

The importance of this contribution is justified by the high popularity of this method in the data analysis process in life sciences and technology. Our speculative implementation, which also includes explicit usage of Intel's SSE and SSE2 media extensions, achieves an impressive speedup over the original SOM-PAK (close 5 on average). This performance adds a valuable extra benefit in this process, allowing scientist and researchers to analyze their data nearly interactively.

References

[1] D. Merkl A. Rauber, P. Tomisch. parSOM: a parallel implementation of the self organizing map exploiting cache effectsmaking the SOM fit for interactive high-performance data analysis. In *Proc. of the IEEE-INNS-ENNS Int. Joint Conf. on Neural Networks*, volume 6, pages 177–182, 2000.

[2] M. Barcena, C. S. Martin, F. Weise, S. Ayora, J. C. Alonso, and J. M. Carazo. Polymorphic quaternary organization of the bacillus subtilis bacteriophage SPP1 replicative helicase (G40 P). *J Mol Biol*, 283:809–819, 1998.

[3] W. Baumeister and A. C. Steven. Macromolecular electron microscopy in the era of structural genomics. *Trends Biochem Sci*, 25:624–31, 2000.

[4] Intel Corpation. Intel C/C++ and Intel Fortran Compilers for Linux. Available at http://www.intel.com/software/products/compilers.

[5] Y. Gomez-Llorente, R. J. Fletcher, X. S. Chen, J. M. Carazo, and C. San Martin. Polymorphism and double hexamer structure in the archaeal helicase mcm from methanobacterium thermoautotrophicum. *Submitted*, 2005.

[6] M. Cottrell J.C. Fort, P. Letremy. Advantages and drawbacks of the batch kohonen algorithm. In *10th European Symp. On Artificial Neural Networks*, Bruges (Belgium), 2005.

[7] Ronald N. Kalla, Balaram Sinharoy, and Joel M. Tendler. IBM Power5 chip: A dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, 2004.

[8] S. Kaski, J. Kangas, and T. Kohonen. Bibliography of self-organizing map (SOM) papers:1981–1997. 1:102–350, 1998.

[9] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. pages 509–521, 1988.

[10] R. D. Lawrence, G. S. Almasi, and H. E. Rushmeier. A scalable parallel algorithm for self-organizing maps with applications to sparse data mining problems. *Data Min. Knowl. Discov.*, 3(2):171–195, 1999.

[11] F. Luengo, A.S. Cofi no, and J.M. Gutierrez. GRID oriented implementation of self-organizing maps for data mining in meteorology, 2004.

[12] R. Marabini and J. M. Carazo. Pattern recognition and classification of images of biological macromolecules using artificial neural networks. *Biophys J*, 66:1804–1814, 1994.

[13] Deborah T. Marr, Frank Binns, David L. Hill, Glenn Hinton, David A. Koufaty, J. Alan Miller, and Michael Upton. Hyper-threading technology architecture and microarchitecture. *INTEL-TECH-J*, 6(1):4–15, February 2002.

[14] H. Y. Ming and N Ahuja. A data partition method for parallel self-organizing map. In *Proc. of the IEEE Int. Joint Conf. on Neural Networks 1999 (IJCNN 99)*, volume 3, pages 1929–33, 1999.

[15] R. D. Pascual-Marqui, A. Pascual-Montano, K. Kochi, and J. M. Carazo. Smoothly distributed fuzzy c-means: a new self-organizing map. *Pattern Recognition*, 34:2395–2402, 2001.

[16] A. Pascual-Montano, L. E. Donate, M. Valle, M. Barcena, R. D. Pascual-Marqui, and J. M. Carazo. A novel neural network technique for analysis and classification of EM single-particle images. *J Struct Biol*, 133:233–245, 2001.

[17] A. Pascual-Montano, K. A. Taylor, H. Winkler, R. D. Pascual-Marqui, and J. M. Carazo. Quantitative self-organizing maps for clustering electron tomograms. *J Struct Biol*, 138:114–122, 2002.

[18] Kohonen T., Hynninen J., Kangas J., and Laaksonen J. SOM PAK: The self-organizing map program package. Technical Report Technical Report A31, Helsinki University of Technology, Laboratory of Computer and Information Science, 1996.

[19] X. Tian, A. Bik, M Girkar, P. Grey, H. Saito, and E. Su. Intel OpenMP C++/Fortran compiler for hyper-threading technology: Implementation and performance. *Intel Technology Journal*, 6(1), 2002.

[20] Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *25 Years ISCA: Retrospectives and Reprints*, pages 533–544, 1998.

[21] H. Winkler and K. A. Taylor. Multivariate statistical analysis of three-dimensional cross-bridge motifs in insect flight muscle. *Ultramicroscopy*, 77:141–152, 1999.